

# NORMAL FORMS

## Chapter 19

in the Course Textbook

Schema Refinement and Normal Forms

ramesh@cs.ubc.ca

<http://www.cs.ubc.ca/~ramesh/cpsc304>

# Normalization Theory

---

*Relational Normalization Theory* - A set of concepts and algorithms that can help with the evaluation and refinement of the designs obtained through the ER approach.

Main tools used for refinement

- The notion of **Functional Dependency**
- To a lesser extent the concept of **Multivalued** and **Join Dependencies**

Tools are used to detect *situations* in which the ER design *UNNATURALLY* places attributes of two distinct entity types into the same relation schema

Characterized as *Normal Forms*

# Normalization

---

## Normalization Theory

- Forces relations into an appropriate *normal form* using **decompositions**
- Break up *unhappy unions* of *unrelated* entity types in schemas - Sounds like ending an unhappy marriage :)

Central role of decompositions - Normalization theory is also called *relational decomposition theory*

# Why Decompose Relations?

---

We need to address the issue of **Redundancy** to answer the question.

- Some information is stored repeatedly in the database  $\Rightarrow$  *REDUNDANCY*
- Redundancy is the main cause of some problems

What problems arise because of redundancy?

- Redundancy gives rise to **ANOMALIES**
  - ▷ **UPDATE ANOMALIES:** If one copy of such repeated data is updated, all copies need to be similarly updated to prevent inconsistency.
  - ▷ **INSERTION ANOMALIES:** It may be impossible to store certain information without storing some other, unrelated information.
  - ▷ **DELETION ANOMALIES:** It may be impossible to delete certain information without losing some other, unrelated information as well.

## EXAMPLE

Let us illustrate anomalies with two examples.

- Consider the following relation and its instance:  
Person(SSN, Name, Address, Hobby)

SSN	Name	Address	Hobby
000001001	Dudley	1 PivetDrive	Bullying
000001001	Dudley	1 PivetDrive	Eating
000001101	Petunia	10 PivetDrive	Cooking
000002011	Dursley	11 PivetDrive	Drinking
000002011	Dursley	11 PivetDrive	Sleeping

- Note that some information is redundant in the instance

## UPDATE ANOMALY

**Update Anomaly:** If Dudley moves to a new place, both tuples containing Dudley have to be updated.

- The red tuples have to be updated.

SSN	Name	Address	Hobby
000001001	Dudley	1 PivetDrive	Bullying
000001001	Dudley	1 PivetDrive	Eating
000001101	Petunia	10 PivetDrive	Cooking
000002011	Dursley	11 PivetDrive	Drinking
000002011	Dursley	11 PivetDrive	Sleeping

## INSERTION ANOMALY

**Insert Anomaly:** Suppose we add Harry Potter to the Person relation instance, but his hobbies are not yet known. What can we do?

- We could add  $\langle 123456789, \text{HarryPotter}, \text{Hogwarts}, \text{NULL} \rangle$
- Hobby is part of the PRIMARY KEY and has to be non-null
- DBs usually build *indices* on PRIMARY KEY to support fast lookup. Indices have to be non-null.

Assuming this problem is solved and we are allowed NULL values, we insert the tuple  $\langle 123456789, \text{HarryPotter}, \text{Hogwarts}, \text{NULL} \rangle$

Suppose a tuple  $\langle 123456789, \text{HarryPotter}, \text{Hogwarts}, \text{Quidditch} \rangle$  needs to be inserted subsequently.

- Should we add this tuple? or
- Should we replace  $\langle 123456789, \text{HarryPotter}, \text{Hogwarts}, \text{NULL} \rangle$  ?

Problem: Redundancy - If Harry Potter is described by ATMOST one tuple, then only one course of action would be possible.

## DELETION ANOMALY

Suppose cooking is no longer the hobby of Petunia. How can this be deleted?

- We could delete the tuple talking about Petunia's cooking hobby.
- PROBLEM: There is only one tuple containing address and SSN for Petunia.
- Perfectly valid information about Petunia is *LOST*
- We could replace Cooking with NULL - this leads to the same problems as before about NULL valued PRIMARY KEY.
- REDUNDANCY is the problem

# ONE DECOMPOSITION FIX

Decompose Person relation into two relations below.

- NewPerson(SSN, Name, Address)
- Hobby(SSN, Hobby)

SSN	Name	Address
000001001	Dudley	1 PivetDrive
000001101	Petunia	10 PivetDrive
000002011	Dursley	11 PivetDrive

SSN	Hobby
000001001	Bullying
000001001	Eating
000001101	Cooking
000002011	Drinking
000002011	Sleeping

## SECOND EXAMPLE

Consider the following relation schema and instance

- Hourly\_Emps(SSN, Name, Lot, Rating, Hourly\_Wages, Hours\_Worked)

SSN	Name	Lot	Rating	Hourly_Wages	Hours_Worked
123 – 22 – 3666	Attishoo	48	8	10	40
231 – 31 – 5368	Smiley	22	8	10	30
131 – 24 – 3650	Smethurst	35	5	7	30
434 – 26 – 3751	Guldu	35	5	7	32
612 – 67 – 4134	Madayan	35	8	10	40

- Key is SSN
- Let us also assume the following *dependency*: The Hourly\_Wages attribute is determined by the Rating attribute i.e., For two tuples, if the Rating values are the same then their corresponding Hourly\_wages values are also the same.
- This Integrity Constraint is an example of a *functional dependency*

## REDUNDANCY

- The rating value 8 corresponds to an Hourly\_Wage of 10 and this is repeatedly stored *thrice*.

# ANOMALIES

## UPDATE ANOMALY

- Suppose that the Hourly\_Wages of Attishoo is modified to be 11. This causes an inconsistency as it does not make a similar change to Smiley and Madayan.
- Violates the Integrity Constraint or the functional dependency.

## INSERTION ANOMALY

- A tuple for an employee cannot be inserted unless the Hourly\_Wages value for the employee's Rating is known.

## DELETION ANOMALY

- Suppose we delete the employees Smethurst and Guldu, we lose the information that Rating value 5 corresponds to the Hourly\_Wages of 7. The correspondence between 5 and 7 is lost here.

## Decomposition to FIX the Anomalies

- The following decomposition fixes the problem in the Hourly\_Emps relation

SSN	Name	Lot	Rating	Hours_Worked
123 – 22 – 3666	Attishoo	48	8	40
231 – 31 – 5368	Smiley	22	8	30
131 – 24 – 3650	Smethurst	35	5	30
434 – 26 – 3751	Guldu	35	5	32
612 – 67 – 4134	Madayan	35	8	40

Rating	Hourly_Wages
8	10
5	7

# DECOMPOSITION

## DECOMPOSITION of a relation schema $R$

- Replacing the relation schema  $R$  by two or more relation schemas that
- each contain a subset of the attributes of  $R$
- and together include all the attributes in  $R$

Store instances in  $R$  by storing the projections of the instance

## Questions one should ask

- Do we need to decompose a relation?
  - ▷ To answer this question, several **normal forms** have been proposed for relations.
  - ▷ Certain properties cannot arise if relations are known to be in a given normal form.
- What problems (if any) does a decomposition cause?
  - ▷ The **lossless-join** property - helps recover original instance from the corresponding smaller instances
  - ▷ The **dependency-preservation** property - Enforce any constraints on the original relation by simply enforcing them on the smaller relations

# FUNCTIONAL DEPENDENCIES

## Functional Dependency (FD)

- R - Relation Schema, X,Y - Nonempty sets of attributes of R
- An instance r of R satisfies the FD  $X \rightarrow Y$  if:  
For all tuples  $t_1, t_2$  in r, If  $t_1.X = t_2.X$ , then  $t_1.Y = t_2.Y$ .
- We say X determines Y or X functionally determines Y.
- $t_1.X$  denotes the projection of tuple  $t_1$  onto the attributes in X
- An FD is an integrity constraint that generalizes the concept of a key.
- In essence, an FD  $X \rightarrow Y$  says that if two tuples agree on the values of X, then they must also agree on the values of Y.

A	B	C	D
$a_1$	$b_1$	$c_1$	$d_1$
$a_1$	$b_1$	$c_1$	$d_2$
$a_1$	$b_2$	$c_2$	$d_1$
$a_2$	$b_1$	$c_3$	$d_1$

$AB \rightarrow C$  in the example.

An FD is not the same as a key constraint.

If two tuples differ in either A or B, they can differ in C attribute.

Adding  $\langle a_1, b_1, c_2, d_1 \rangle$  violates the FD.

A legal instance of a relation must satisfy all ICs including all specified FDs.

# Functional Dependencies - Points to Keep in Mind

- Integrity Constraints - Identified and Specified based on the semantics of the real world enterprise being modelled.
- Hence, Functional Dependencies are *specified* and *identified* when modelling.
- Given a relation, we can say that a certain FD does not hold.
  - ▷ All we need to do is to produce two tuples which contradict the FD.
- Looking at an instance of a relation, we cannot deduce that an FD DOES HOLD. Why?
  - ▷ An FD (like any other IC) is a statement about ALL possible legal instances of the relation.
  - ▷ Verifying ALL instances of a relation is difficult.
- PRIMARY KEY constraint - Special case of FD. Attributes in the key is  $X$  and the set of all attributes is  $Y$ . The FD does not require  $X$  to be minimal.
- If  $X \rightarrow Y$  holds, where  $Y$  is the set of *all* attributes, and there is a  $V \subset X$  such that  $V \rightarrow Y$  holds, then  $X$  is a *superkey*.

## Reasoning about FDs

- Given a set of FDs  $F$  that hold over a relational schema  $R$ ,  
We can *infer* additional FDs from the given set of FDs.
- An FD  $f$  is **implied by**  $F$  if  $f$  also holds on **every** relational instance whenever  $F$  holds.
- Note:  $f$  should hold on **every** instance and not just *any* instance.
- We can now talk about the *set of all FDs that can be inferred from*  $F$
- This is called the **CLOSURE** of the set of FDs  $F$ .

# CLOSURE of FDs

F - Given set of FDs

**Closure** of F - denoted by  $F^+$

- Set of all FDs that are implied by the FDs in F

How can we **compute** this closure?

- Rules of inference (Just like propositional calculus)
- Three rules of inference which can be applied repeatedly to compute the closure.
- The rules are called **Armstrong's Axioms**

$X \rightarrow Y$  is a TRIVIAL FD whenever  $Y \subseteq X$ .

## Armstrong's Axioms

R - Relation Schema, X, Y, Z - Sets of attributes over R

- **Reflexivity:** If  $Y \subseteq X$ , then  $X \rightarrow Y$  (**TRIVIAL FD**)
- **Augmentation:** If  $X \rightarrow Y$ , then  $XZ \rightarrow YZ$  for any Z
- **Transitivity:** If  $X \rightarrow Y$  and  $Y \rightarrow Z$ , then  $X \rightarrow Z$

Theorem: Armstrong's Axioms are **sound** and **complete**

- **Soundness:** The axioms generate only FDs in  $F^+$
- **Completeness:** Every FD in  $F^+$  is generated by repeated application of the axioms

For convenience, there are the following additional rules:

- **Union:** If  $X \rightarrow Y$  and  $X \rightarrow Z$ , then  $X \rightarrow YZ$
- **Decomposition:** If  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$

The additional rules are not essential and can be proved using the basic three axioms.

# Examples

- Consider a relational schema  $R(A, B, C)$  and let  $F$  be the set of FDs  $A \rightarrow B$  and  $B \rightarrow C$
- Reflexivity gives all the *trivial FDs*:  $X \rightarrow Y$  where  $Y \subseteq X$ ,  $X \subseteq ABC$  and  $Y \subseteq ABC$
- From transitivity, we get  $A \rightarrow C$ .
- Augmentation gives the following.
  - ▷  $A \rightarrow B$  and  $C$  gives,  $AC \rightarrow BC$ .
  - ▷  $A \rightarrow C$  and  $B$  gives,  $AB \rightarrow BC$ .
  - ▷  $B \rightarrow C$  and  $A$  gives,  $AB \rightarrow AC$ .
- Union of  $A \rightarrow B$  and  $A \rightarrow C$  gives  $A \rightarrow BC$

# Attribute Closure

- Computing ALL FDs in  $F^+$  is exponential.
- However, checking whether a specified dependency  $X \rightarrow Y$  is in the closure of  $F$  can be done efficiently without computing  $F^+$ .
- All we need to compute is the *attribute closure*  $X^+$  with respect to  $F$ .
  - ▶ The set of attributes  $A$  such that  $X \rightarrow A$  can be inferred from Armstrong's Axioms.

## Algorithm Compute Attribute Closure

**Given:**  $F, X$

**Output:**  $X^+$

$X^+ = \{X\}$

Repeat until nothing new is added to  $X^+ \{$

For  $U \rightarrow V$  in  $F$  such that  $U \subseteq X$

$X^+ = X^+ \cup V$

}

## Attribute Closure Continued

- Algorithm to compute attribute closure can be modified to find keys.
- Start with attribute set  $X$ . Stop as soon as the closure contains *all* the attributes in the relation schema.
- By varying  $X$  and the order of considering FDs, we can obtain all candidate keys.

## Example I

- Problem:  $R(A, B, C, D)$  is the relation schema.  $F = \{AB \rightarrow C, C \rightarrow D, D \rightarrow A\}$

- What are the non-trivial FDs that follow from  $F$ ? (Restrict to single attributes on the right hand side)

**Ans:** Consider attribute closures.

Closure	New FDs
$A^+ = A$	None
$B^+ = B$	None
$C^+ = ACD$	$C \rightarrow A$
$D^+ = AD$	None
$AB^+ = ABCD$	$AB \rightarrow D$
$AC^+ = ABCD$	$AC \rightarrow D$
$AD^+ = ABCD$	None

Closure	New FDs
$BC^+ = ABCD$	$BC \rightarrow A, BC \rightarrow D$
$BD^+ = ABCD$	$BD \rightarrow A, BD \rightarrow C$
$CD^+ = ACD$	$CD \rightarrow A$
$ABC^+ = ABCD$	$ABC \rightarrow D$
$ABD^+ = ABCD$	$ABD \rightarrow C$
$ACD^+ = ACD$	None
$BCD^+ = ABCD$	$BCD \rightarrow A$

- What are the *keys* of  $R$ ?

**Ans:** The **red** attribute sets are the ones whose closure have the entire set of attributes. The minimal sets from these red sets are  $AB, BC$  and  $BD$  which are the keys.

- Superkeys of  $R$ ? **Ans:**  $ABC, ABD, BCD, ABCD$

## Example II

- Problem:  $S(A, B, C, D)$  is the relation schema.  $F = \{A \rightarrow B, B \rightarrow C, B \rightarrow D\}$

- What are the non-trivial FDs that follow from  $F$ ? (Restrict to single attributes on the right hand side)

**Ans:** Consider attribute closures.

Closure	New FDs
$A^+ = ABCD$	$A \rightarrow C, A \rightarrow D$
$B^+ = BCD$	None
$C^+ = C$	None
$D^+ = D$	None
$AB^+ = ABCD$	$AB \rightarrow C, AB \rightarrow D$
$AC^+ = ABCD$	$AC \rightarrow B, AC \rightarrow D$
$AD^+ = ABCD$	$AD \rightarrow B, AD \rightarrow C$

Closure	New FDs
$BC^+ = BCD$	$BC \rightarrow D$
$BD^+ = BCD$	$BD \rightarrow C$
$CD^+ = CD$	None
$ABC^+ = ABCD$	$ABC \rightarrow D$
$ABD^+ = ABCD$	$ABD \rightarrow C$
$ACD^+ = ABCD$	$ACD \rightarrow B$
$BCD^+ = BCD$	None

- What are the *keys* of  $R$ ?

**Ans:** The **red** attribute sets are the ones whose closure have the entire set of attributes. The minimal set from these red sets is  $A$  which is the only key for  $R$ .

- Superkeys of  $R$ ? **Ans:**  $AB, AC, AD, ABC, ABD, ACD, ABCD$

## EXAMPLE SCHEMAS FOR REFERENCE

- Contracts(contractid, supplierid, projectid, deptid, partid, qty, value) - denoted as Contracts(CSJDPQV)
- Hourly\_Emps(ssn, name, lot, rating, hourly\_wages, hours\_worked) - denoted as Hourly\_Emps(SNLRWH)
- Reserves(sid, bid, day, credit\_card) denoted as Reserves(SBDC)

# NORMAL FORMS

First Normal Form (1NF): Equivalent to the definition of relational model. Included here for completeness

- A relational schema is in 1NF if and only if the domains of all attributes of  $R$  are atomic (A domain is atomic if elements of the domain are considered to be indivisible units). Therefore, attributes cannot be set-valued (which is part of the definition of the relational model)

Second Normal Form (2NF): Of historical interest. Definition included for completeness.

- **Partial dependency:** An FD  $X \rightarrow Y$  is said to be a *partial dependency* if there is a  $Z \subset X$  such that  $Z \rightarrow Y$ . We say  $Y$  is partially dependent on  $X$ .
- **Second Normal Form (2NF):** A relation schema  $R$  is in 2NF if each attribute  $A$  in  $R$  satisfies one of the following criteria:
  1.  $A$  is part of a candidate key.
  2.  $A$  is not *partially* dependent on a candidate key.

## 2NF - No partial key dependencies

- If  $R$  is in 2NF, then it is true that: For every FD  $X \rightarrow Y$  where  $X$  is a key and  $Y$  is a non-key attribute, then no proper subset of  $X$  determines  $Y$ .
  - ▶ i.e.,  $\nexists Z \subset X$  such that  $Z \rightarrow Y$ .

## Third Normal Form (3NF)

$R$  - Relation Schema,  $F$  - Set of FDs given to hold over  $R$ ,  $X$  - A subset of attributes of  $R$ ,  $A$  - a single attribute of  $R$

$R$  is said to be in Third Normal Form if for *every* FD  $X \rightarrow A$ , one of the following statements is true:

- $A \in X$  i.e. the FD is trivial
- $X$  is a superkey
- $A$  is a part of some key for  $R$

A key for the relation is a minimal set of attributes that uniquely determines all other attributes.  $A$  has to be part of *any* key.

Finding all keys of a relation schema is known to be NP-Complete. So is the problem of determining whether a relation schema is in 3NF.

## 3NF - A closer look

Suppose  $X \rightarrow A$  violates 3NF - There are two cases when this can happen.

- $X \subset K$  where  $K$  is a key. This is precisely a *partial dependency*, as  $K \rightarrow A$ . Thus  $(X, A)$  is stored redundantly in this case.
- $X$  is not a proper subset of *any* key. This is sometimes called a *transitive dependency*. Thus we have a chain of dependencies  $K \rightarrow X \rightarrow A$ .
  - ▷ We cannot associate an  $X$  value with a  $K$  value unless we also associate an  $A$  value with an  $X$  value.
- In both cases attribute  $A$  is not part of any key.

Some redundancy is still possible in 3NF.

- $X \rightarrow A$  - A nontrivial dependency and  $X$  is not a superkey,  $A$  is part of a key.
- In this case problems with partial and transitive dependencies persist.

## Example of 3NF

Consider the relation  $\text{Reserves}(S, B, D, C)$ ,  $S$  - Sailor ID,  $B$  - Boat ID,  $D$  - Day,  $C$  - Credit Card

FD  $S \rightarrow C$  which states that a sailor is associated with a unique credit card to pay for reservations. (*A partial dependency*)

- The only key is  $SBD$ .
- $S$  is not a key.  $C$  is not part of a key. Hence this relation is not in 3NF
- Hence  $(S, C)$  pairs are stored redundantly, one for each reservation made by a sailor.
- Suppose we know  $C \rightarrow S$ , i.e., Every credit card is uniquely associated with its owner (a sailor).
- This means that  $CBD$  is a *key* for the relation  $\text{Reserves}$ . In this case,  $S \rightarrow C$  does not violate the conditions of 3NF. In this case the relation is in 3NF.
- Nonetheless,  $(S, C)$  is still redundantly stored for all tuples containing the same  $S$  value.

## Boyce-Codd Normal Form (BCNF)

$R$  - Relation Schema,  $F$  - Set of FDs given to hold over  $R$ ,  $X$  - A subset of attributes of  $R$ ,  $A$  - a single attribute of  $R$

$R$  is said to be in Boyce-Codd Normal Form if for *every* FD  $X \rightarrow A$ , one of the following statements is true:

- $A \in X$  i.e. the FD is trivial
- $X$  is a superkey

Intuitively, the only nontrivial dependencies that hold are those in which a key determines some attributes.

BCNF ensures that no redundancy can be detected using FDs alone. Thus most desirable in terms of redundancy.

# BCNF

- A BCNF schema can have more than one key.
- Example:  $R = (ABCD, F)$  where  $F = \{AB \rightarrow CD, AC \rightarrow BD\}$ .
- There are two keys  $AB$  and  $AC$ . This schema is in BCNF.
- Important property of BCNF : Instances do not contain redundant information.
- Example:

A	B	C	D
1	1	3	4
2	1	3	4

Does this instance store redundant information?

Two tuples are the same in all but one attribute. LOOKS LIKE IT..

But having almost identical tuples does not constitute redundancy.

- Redundancy: When values of some set of attributes  $X$  necessarily imply a value that must exist in  $A$  (FD)
- Redundancy is eliminated if we store this association  $(X, A)$  exactly once in a separate relation.  $R$  however does not have any FDs over  $BCD$  - Hence no redundancy.

# BCNF

- A DBMS automatically eliminates one type of redundancy. Two tuples with same key values are forbidden.
- BCNF precludes associations that do not contain keys. So relations over BCNF schemas do not have any redundancy.
- As a result, No DELETE and UPDATE anomalies. However, INSERTION anomalies can occur (when more than one key is present)

A	B	C	D
1	1	3	4
2	1	3	4
3	4	NULL	5
3	NULL	2	5

Associations over ABD and ACD are added

Attribute C over association ABD  $\rightarrow$  C is unknown and hence NULL in tuple 3

Attribute D over association ABC  $\rightarrow$  D is unknown and hence NULL in tuple 4

- We cannot say whether the two tuples are the same or not.
- SQL standard - designate ONE PRIMARY KEY and prohibit NULL values in its attributes.

# BCNF Versus 3NF

- Redundancy: BCNF has NONE while 3NF still has some redundancy
- The intrinsic merit of the extra condition in 3NF: Nothing great about the extra third condition.
- Why is 3NF useful then?
  - ▷ Nice algorithmic properties.
  - ▷ Excepting certain dependencies on key attributes, **Every relation schema can be decomposed into a collection of 3NF relations using only decompositions that have desirable properties**
  - ▷ Such a guarantee does not exist for BCNF relations.
  - ▷ 3NF weakens BCNF just enough to let this guarantee hold.
  - ▷ 3NF or even a non-3NF schema is thus a COMPROMISE

## A Remark about Normal Forms

Note that a Relation schema  $R$  satisfies:

Schema $R$ in <i>in</i>	<i>is also in</i>
BCNF	3NF
3NF	2NF
2NF	1NF

Progression of Normalization theory - incremental.

# Decompositions - Desirable Properties

Lossless-Join Decomposition

Dependency-Preserving Decomposition

# Lossless-Join Decomposition

- R - Relation Schema, F - Set of FDs over R
- Decomposition of R into *two* schemas with attribute sets X and Y is said to be a **lossless-join decomposition with respect to F** if
  - ▷ For every instance r of R that satisfies the dependencies in F,  $\pi_X(r) \bowtie \pi_Y(r) = r$ .
- We can recover the original relation from the decomposed relations.
- Can be extended to cover decomposition into more than two schemas.
- $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$  is straightforward to comprehend. The other direction is not so direct.

# Example

- The following example illustrates this:

Instance $r$	$\pi_{SP}(r)$	$\pi_{PD}(r)$	$\pi_{SP}(r) \bowtie \pi_{PD}(r)$																												
			<table border="1" style="border-collapse: collapse; text-align: center;"> <thead> <tr> <th>S</th> <th>P</th> <th>D</th> </tr> </thead> <tbody> <tr> <td><math>s_1</math></td> <td><math>p_1</math></td> <td><math>d_1</math></td> </tr> <tr> <td><math>s_2</math></td> <td><math>p_2</math></td> <td><math>d_2</math></td> </tr> <tr> <td><math>s_3</math></td> <td><math>p_1</math></td> <td><math>d_3</math></td> </tr> <tr> <td style="color: red;"><math>s_1</math></td> <td style="color: red;"><math>p_1</math></td> <td style="color: red;"><math>d_3</math></td> </tr> <tr> <td style="color: red;"><math>s_3</math></td> <td style="color: red;"><math>p_1</math></td> <td style="color: red;"><math>d_1</math></td> </tr> </tbody> </table>	S	P	D	$s_1$	$p_1$	$d_1$	$s_2$	$p_2$	$d_2$	$s_3$	$p_1$	$d_3$	$s_1$	$p_1$	$d_3$	$s_3$	$p_1$	$d_1$										
S	P	D																													
$s_1$	$p_1$	$d_1$																													
$s_2$	$p_2$	$d_2$																													
$s_3$	$p_1$	$d_3$																													
$s_1$	$p_1$	$d_3$																													
$s_3$	$p_1$	$d_1$																													
<table border="1" style="border-collapse: collapse; text-align: center;"> <thead> <tr> <th>S</th> <th>P</th> <th>D</th> </tr> </thead> <tbody> <tr> <td><math>s_1</math></td> <td><math>p_1</math></td> <td><math>d_1</math></td> </tr> <tr> <td><math>s_2</math></td> <td><math>p_2</math></td> <td><math>d_2</math></td> </tr> <tr> <td><math>s_3</math></td> <td><math>p_1</math></td> <td><math>d_3</math></td> </tr> </tbody> </table>	S	P	D	$s_1$	$p_1$	$d_1$	$s_2$	$p_2$	$d_2$	$s_3$	$p_1$	$d_3$	<table border="1" style="border-collapse: collapse; text-align: center;"> <thead> <tr> <th>S</th> <th>P</th> </tr> </thead> <tbody> <tr> <td><math>s_1</math></td> <td><math>p_1</math></td> </tr> <tr> <td><math>s_2</math></td> <td><math>p_2</math></td> </tr> <tr> <td><math>s_3</math></td> <td><math>p_1</math></td> </tr> </tbody> </table>	S	P	$s_1$	$p_1$	$s_2$	$p_2$	$s_3$	$p_1$	<table border="1" style="border-collapse: collapse; text-align: center;"> <thead> <tr> <th>P</th> <th>D</th> </tr> </thead> <tbody> <tr> <td><math>p_1</math></td> <td><math>d_1</math></td> </tr> <tr> <td><math>p_2</math></td> <td><math>d_2</math></td> </tr> <tr> <td><math>p_1</math></td> <td><math>d_3</math></td> </tr> </tbody> </table>	P	D	$p_1$	$d_1$	$p_2$	$d_2$	$p_1$	$d_3$	
S	P	D																													
$s_1$	$p_1$	$d_1$																													
$s_2$	$p_2$	$d_2$																													
$s_3$	$p_1$	$d_3$																													
S	P																														
$s_1$	$p_1$																														
$s_2$	$p_2$																														
$s_3$	$p_1$																														
P	D																														
$p_1$	$d_1$																														
$p_2$	$d_2$																														
$p_1$	$d_3$																														

- By replacing  $r$  with the decomposition, we *lose* information. We cannot say whether the **red** tuples do not hold.

## Lossless-Join Decomposition

All decompositions used to eliminate redundancy **must** be lossless. The following test is useful.

- **Theorem:** Let  $R$  be a relation and  $F$  be a set of FDs that hold over  $R$ . The decomposition of  $R$  into relations with attribute sets  $R_1$  and  $R_2$  is lossless if and only if  $F^+$  contains either the FD  $R_1 \cap R_2 \rightarrow R_1$  or the FD  $R_1 \cap R_2 \rightarrow R_2$ .
- That is, attributes common to  $R_1$  and  $R_2$  must contain a key for either  $R_1$  or  $R_2$ .

If a relation is decomposed into more than two relations, an efficient algorithm (polynomial in the size of the dependency set) exists to test whether or not the decomposition is lossless.

## Example

- Consider the Hourly\_Emps relation containing the attributes S - ssn, N - Name, L - Lot, R - Rating, W - Hourly wages and H - Hours Worked.
- The FD  $R \rightarrow W$  causes a violation of 3NF.
- We decomposed the relation into SNLRH and RW. Note that R is common to both the decomposed relations and  $R \rightarrow W$  holds.
- For the relation RW, R is the key. Thus this decomposition is lossless-join.
- A general observation: *If an FD  $X \rightarrow Y$  holds over a relation R and  $X \cap Y$  is empty, the decomposition of R into  $R - Y$  and XY is lossless.*
- Another observation: *Suppose that R is decomposed into  $R_1$  and  $R_2$  through a lossless-join decomposition and  $R_1$  is further decomposed into  $R_{11}$  and  $R_{12}$  through another lossless join decomposition. Then the decomposition of R into  $R_{11}$ ,  $R_{12}$  and  $R_2$  is lossless-join.*

## Dependency-Preserving Decomposition

- Let us first consider Contracts( $CSJDPQV$ ). The given FDs are  $C \rightarrow CSJDPQV$ ,  $JP \rightarrow C$  and  $SD \rightarrow P$ .
- $SD$  is not a key. Hence  $SD \rightarrow P$  violates BCNF.
- Decompose  $CSJDPQV$  into  $CSJDQV$  and  $SDP$  to address this violation,  $SD$  being the key for the second relation. This is lossless-join.
- Consider an Integrity Constraint  $JP \rightarrow C$ .
- Before decomposition,  $JP \rightarrow C$  can be enforced in the *same* relation. However, once the decomposition takes place, we need to *JOIN*  $CSJDQV$  and  $SDP$  in order to enforce this IC, which may be expensive.
- This decomposition is *NOT* dependency-preserving.
- Intuitively, *a dependency-preserving decomposition allows us to enforce all FDs by examining a single relation instance on each insertion or modification of a tuple.*

# Dependency-Preserving Decomposition

## Projection of FDs on an Attribute Set

- R - Relation schema that is decomposed into two schemas with attribute sets X and Y
- F - Set of FDs on R
- **Projection of F on X:** The set of FDs in  $F^+$  that involve only attributes in X. This is denoted as  $F_X$
- An FD  $U \rightarrow V$  in  $F^+$  is in  $F_X$  only if *all* the attributes in U and V are in X

## Dependency-Preserving Decomposition:

- The decomposition of a relation schema R with FDs F into schemas with attribute sets X and Y is **dependency-preserving** if  $(F_X \cup F_Y)^+ = F^+$ .

# Dependency-Preserving Decomposition

- R is a relation with attributes ABC and  $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$
- Suppose R is decomposed into relations with attributes AB and BC - Is this dependency-preserving?
- $A \rightarrow B$  is in  $F_{AB}$  and  $B \rightarrow C$  is in  $F_{BC}$ .
- What happens to  $C \rightarrow A$ ?
- We consider  $F^+$ . The closure also contains  $C \rightarrow B$ ,  $B \rightarrow A$  and  $A \rightarrow C$ .
- Hence  $F_{AB}$  also contains  $B \rightarrow A$  and  $F_{BC}$  also contains  $C \rightarrow B$ .
- If we consider  $F_{AB} \cup F_{BC}$ , the closure of this union indeed implies  $C \rightarrow A$ .
- Hence this decomposition preserves dependency.

# Normalization

- We now consider *algorithms* for converting relations into BCNF and 3NF.
- If a relation is not in BCNF, it is possible to obtain a lossless-join decomposition into a collection of relation schemas in BCNF.
  - ▷ Unfortunately, *dependency-preservation* is not guaranteed.
- There is *always* a dependency-preserving, lossless-join decomposition of a relation into a collection of 3NF schemas.

# Decomposition into BCNF

## Algorithm:

- **Given:**  $R$  - Relation Schema,  $F$  - a set of FDs
- Let  $X \rightarrow A$  be the FD that violates BCNF, where  $X \subset R$  and  $A$  is a single attribute of  $R$ . Decompose  $R$  into  $R - A$  and  $XA$ .
- If  $R - A$  or  $XA$  are not in BCNF, continue the decomposition recursively.
- Repeat until no FDs violate BCNF.

## Remarks:

- $X \rightarrow A$  violates BCNF. This means that  $X \rightarrow A$  is not trivial and hence  $A \notin X$ .
- Hence each decomposition carried out in the algorithm is a lossless-join decomposition.
- The process terminates as each decomposition results in relations with strictly fewer attributes.
- Joining instances of the relations obtained yields precisely the corresponding instance of the original relation.

## Example

- R - Relation with attributes CSJDPQV.  $F = \{JP \rightarrow C, SD \rightarrow P\}$ . The key for R is C.
- The relation is not in BCNF. Suppose we start with  $SD \rightarrow P$ . We decompose to get CSJDQV and SDP
- SDP is in BCNF (SD is the key). If we have the additional constraint that  $J \rightarrow S$ . Then CSJDQV is not in BCNF. This would lead to the decomposition CJDQV and JS.
- Thus the decomposition CJDQV, JS and SDP is a lossless-join decomposition of R into BCNF. The steps can be visualized as a tree.
- Suppose we start with  $J \rightarrow S$ . The relation would be decomposed into CJDPQV and JS. The only dependencies that hold over CJDPQV are  $JP \rightarrow C$  and the key dependency  $C \rightarrow CJDPQV$ . JP is a key. Hence the decomposed relations are in BCNF.

# Redundancy

- The decomposition of  $CSJDPQV$  into  $SDP$ ,  $JS$  and  $CJDQV$  is NOT *dependency-preserving*.
- $JP \rightarrow C$  cannot be enforced in a single relation without a join.
- An approach to deal with this is to add a relation  $CJP$ , involving some redundancy.
- Each of  $CJP$ ,  $SDP$ ,  $JS$  and  $CJDQV$  are in BCNF - Some redundancy can be predicted by the FDs.
- Within a single relation there is no redundancy but *across* relations, redundancy may still occur.

## BCNF and Dependency-Preservation

- Sometimes there is NO decomposition into BCNF that is dependency-preserving.
- Example: Relation SBD denoting sailor S has reserved boat B on date D.
- If we have FDs  $SB \rightarrow D$  and  $D \rightarrow B$ , SBD is not in BCNF as D is not a key.
- However, if we decompose it, we cannot preserve the dependency  $SB \rightarrow D$ .

## Decomposition Into 3NF

- Same procedure for BCNF works for 3NF - we can stop early if the 3NF conditions are satisfied.
- However, the resulting decomposition is not necessarily *dependency-preserving*.
- A simple modification can yield a dependency preserving decomposition.
- Instead of FDs  $F$  use a *minimal covers* for  $F$ .

## Minimal Cover for a set of FDs

- A **minimal cover** for a set  $F$  of FDs is a set  $G$  of FDs such that:
  - ▷ Every FD in  $G$  is of the form  $X \rightarrow A$  where  $A$  is a single attribute.
  - ▷  $F^+ = G^+$
  - ▷ For  $H \subset G$ ,  $H^+ \neq F^+$ .
- Every dependency is as small as possible (each attribute on the left side of any dependency is necessary)
- Every dependency in the minimal cover is required for the closure to be equal to  $F^+$

# Method and Example

- The algorithm has the following steps:
  - ▷ Put the FDs in standard form (with 1 attribute on the right hand side)
  - ▷ Minimize Left hand side if possible
  - ▷ Remove redundant FDs
- Consider the dependencies:
- $A \rightarrow B$ ,  $ABCD \rightarrow E$ ,  $EF \rightarrow GH$ ,  $ACDF \rightarrow EG$
- A minimal cover for the above dependencies is:
- $A \rightarrow B$ ,  $ACD \rightarrow E$ ,  $EF \rightarrow G$  and  $EF \rightarrow H$ .

## Refining an ER Diagram

- FDs can help refine relations constructed out of an ER Model.
- An ER diagram can express only FDs that determine *all* attributes of a relation (which are essentially key constraints)
- Consider an example:
  - ▷ Employees - Entity with Attributes *ssn*, *name* and *lot* (*ssn* is the key).
  - ▷ Departments - Entity with Attributes *did*, *dname* and *budget* (*did* is the key)
  - ▷ Works\_In is a relationship between Employees and Departments with a descriptive attribute *since*.
  - ▷ Each employee can work in at most one department and hence the relationship is one to many from employees to departments.
- Under the key constraint, the following two relations are modelled.
  - ▷ Workers(*ssn*, *name*, *lot*, *since*)
  - ▷ Departments(*did*, *dname*, *budget*)

## Refining an ER Diagram

- Both Employees and Works\_In are mapped to the same relation Workers.
- Suppose all workers in the same department are assigned the same parking lot, then  $did \rightarrow lot$  is a functional dependency that models this situation.
- There is redundancy in the Workers relation. This can be eliminated by the following decomposition.
  - ▷ Workers2(ssn, name, did, since)
  - ▷ DepLot(did, lot)
- Instead of updating all tuples with the same did, we update only one tuple in the DepLot relation.
- Observe that the Departments relation and DepLot relation have the same key and describe the same entity. Hence we can move lot as an attribute of the Departments relation and have the following decomposition.
  - ▷ Workers(ssn, name, did, since)
  - ▷ Departments(did, dname, budget, lot)
- It seems intuitive to associate lot with employees but the ICs reveal that lot is in fact associated with departments

# Summary

- If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good heuristic.
- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  - ▷ Must consider whether all FDs are preserved. If a lossless-join, dependency preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.
  - ▷ Decompositions should be carried out and/or re-examined while keeping performance requirements in mind.